Homework 5: Trial and error methods

I. Lý thuyết phương pháp

Phương pháp quay lui (backtracking)

- **Ý tưởng:**
  - Theo nguyên tắc vét cạn, nhưng chỉ xét những trường hợp khả quan.
  - Dùng để giải bài toán liệt kê các cấu hình:
    - Mỗi cấu hình được xây dựng bằng cách xác định từng phần tử.
    - Mỗi phần tử được chọn bằng cách thử các khả năng có thể.
  - Tại mỗi bước, nếu có một lựa chọn được chấp thuận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo. Còn ngược lại không có lựa chọn nào thích hợp thì quay lại bước trước => quay lui.

- **Lược đồ:**

```
Try(i) ≡ //Sinh thành phần thứ i của cấu hình
    for (v thuộc tập khả năng thành phần nghiệm xi)
        if ( xi chấp nhận được giá trị v)
            xi = v;
            <Ghi nhận trạng thái chấp nhận v>;
            if (i = n) //đủ n thành phần của cấu hình đã xác định
                <ghi nhận nghiệm>;
            else //lời gọi sinh thành phần tiếp theo của cấu hình
                Try (i + 1)
            <Khôi phục trạng thái chưa chấp nhận v>;
        endif;
    endfor
End.
```

II. Lập trình. *(trong thư mục part 2 đi kèm)*

Chương trình tóm tắt cho các bài trong part 2

1. Bài toán khớp xâu (stringMatching)

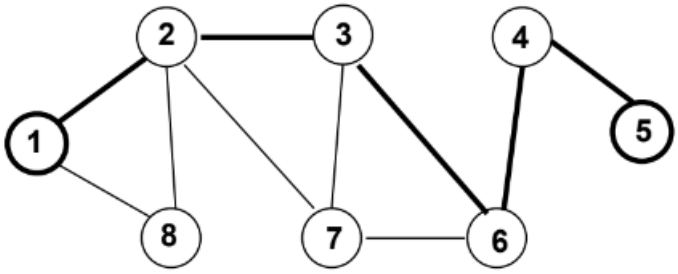
```
def stringMatching(string1, string2):
    lenString1 = len(string1)
    lenString2 = len(string2)
    for i in range(lenString1 - lenString2):
        j = 0
        while j < lenString2 and string2[j] == string1[i + j]:
            j += 1
        if j == lenString2:
            return i
    return -1
```

2. Bài toán liệt kê số nhị phân có độ dài n (ListBinarySequence)

```
n = 3
binary = []
def listBinarySequence(i):
  for bit in range(2):
    binary.append(bit)
    if i == n - 1:
      print(binary)
    else:
      listBinarySequence(i + 1)
    binary.pop()

listBinarySequence(0)
```

3. Bài toán tìm đường trong mê cung (PathInlabyrinth)




```
def pathInLabyrinth(i):
    for v in range(n):
        if labyrinthMatrix[path[i - 1]][v] == 1 and (not passed[v]):
            path.append(v)
            passed[v] = True
            if path[i] == finish:
                print([vertex + 1 for vertex in path])
            else:
                pathInLabyrinth(i + 1)
            passed[v] = False
            path.pop()

path.append(start)
passed[start] = True
pathInLabyrinth(1)
```

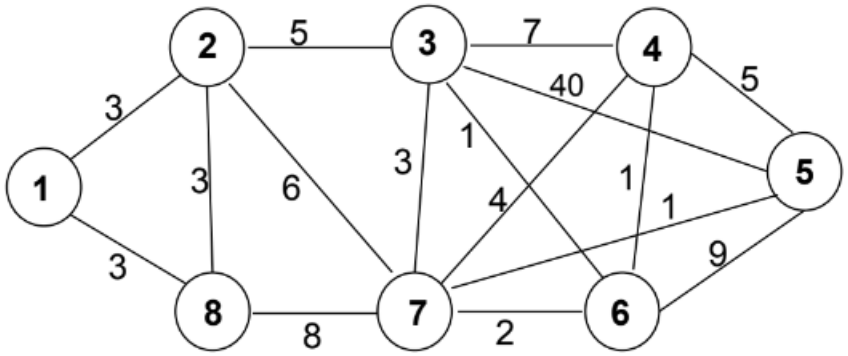
4. Bài toán phân công công việc (WorkAssignment)

	1	2	3	4
A	**5**	**2**	**17**	**8**
B	**6**	**6**	**3**	**7**
C	**5**	**8**	**5**	**10**
D	**7**	**1**	**9**	**4**

```
def workAssignment(i):
    global sMax
    for work in range(n):
        if (not used[work] and (worksAndPeople[i - 1][work] != -1)):
            wayAssigning.append(worksAndPeople[i - 1][work])
            used[work] = True
            if (i == n):
                if sum(wayAssigning) > sMax:
                    sMax = sum(wayAssigning)
                    print("\nSmax is:", sMax)
                    print("way assigning is:", wayAssigning)
            else:
                workAssignment(i + 1)
            used[work] = False
            wayAssigning.pop()

workAssignment(1)
```

5. Bài toán người bán hàng (Salesman)




```
def travelingSalesman(i, c):
    global bestCost
    for city in range(n):
        costPassCity = nCitiesMatrix[path[i - 1]][city]
            if i == n:
            passed[start] = False
        elif i < n:
            passed[start] = True
        if ((costPassCity < INF) and (not passed[city])):
            c1 = c + costPassCity
            if (c1 < bestCost):
                path.append(city)
                passed[city] = True
                if (i == n) and (path[i] == start):
                    bestCost = c1
                    print("\nBest cost is:", bestCost)
                    print("Path is:", [city+1 for city in path])
                elif i <= n - 1:
                    travelingSalesman(i + 1, c1)
```

```
            passed[city] = False
            path.pop()

path.append(start)
passed[start] = True

travelingSalesman(1, cost)
```

III. Đặt bài toán, thiết kế, phân tích và triển khai thuật toán.

- **Đặt bài toán:**

  Bài toán liệt kê các tổ hợp k phần tử của các số từ 1 đến n. **Sử dụng phương pháp backtracking**

  - Input: phần tử n, phần tử k
  - Output: các tổ hợp chập k của n

- **Phân tích bài toán:**
  - Dãy các số ($x_1x_2...x_n$) trong đó x_i = 1,2,...n
  - Dùng giải thuật combinations(i) để sinh giá trị x_i
  - Nếu i = k -1 (tức đã có đủ tổ hợp chập k của n) thì in giá trị nghiệm, ngược lại sinh tiếp x_{i+1} bằng combinations(i+1)

- **Lược đồ:**

Combinations(i) ≡ // sinh số thứ i trong combination

for (num=0..n-1)

if ((i = 0) or (combination[i-1] < num)) // num chấp nhận được

$combination_i$ = num; // ghi nhận trạng thái đã chọn num

if (i = k -1)

printResult(combination);

else:

combination(i+1);

<khôi phục trạng thái chưa chọn num>

endif;

endfor;

End.

Độ phức tạp thuật toán O(n ^ k)

- **Chương trình minh họa** *(trong thư mục part 3 đi kèm)*